Roll No :- 33252 Assignment No :- 3

```
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function to find the index of the process with the earliest arrival time
int find_earliest_arrival(int arr[][3], int n) {
    int min_index = 0;
    for (int i = 1; i < n; i++) {
        if (arr[i][1] < arr[min_index][1]) {
            min_index = i;
        }
    }
    return min_index;
}

// Function to sort processes by burst time
void sort_by_burst_time(int arr[][3], int n) {
    for (int i = 1; i < n - 1; i++) {
        for (int j = 1; j < n - i; j++) {
            if (arr[j][2] > arr[j + 1][2]) {
                for (int k = 0; k < 3; k++) {
                    swap(&arr[j][k], &arr[j + 1][k]);
                }
            }
        }
    }
}

void completion_time(int arr[][3], int n) {
    int completion[n];
    completion[0] = arr[0][1] + arr[0][2];
    for (int i = 1; i < n; i++) {
        if (completion[i - 1] < arr[i][1]) {
            completion[i] = arr[i][1] + arr[i][2];
        } else {
            completion[i] = completion[i - 1] + arr[i][2];
        }
        printf("%d\t\t%d\n", arr[i][0], completion[i]);
    }
}

void sjf_non_preemptive(int arr[][3], int n) {
```

```
    // Find and print the process with the earliest arrival time
    int earliest_index = find_earliest_arrival(arr, n);
    printf("Process with earliest arrival time:\n");
    printf("Process ID: %d, Arrival Time: %d, Burst Time: %d\n",
           arr[earliest_index][0], arr[earliest_index][1], arr[earliest_index][2]);

    // Swap the earliest process to the first position
    if (earliest_index != 0) {
        for (int k = 0; k < 3; k++) {
            swap(&arr[0][k], &arr[earliest_index][k]);
        }
    }

    // Sort the remaining processes by burst time (starting from index 1)
    sort_by_burst_time(arr, n);

    printf("Process ID\tArrival Time\tBurst Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\n", arr[i][0], arr[i][1], arr[i][2]);
    }

    int waiting_time[n];
    int total_waiting_time = 0;
    float avg_time = 0;
    waiting_time[0] = 0;

    printf("Non-Preemptive Waiting Time: \n");
    printf("%d\n", waiting_time[0]);

    for (int i = 1; i < n; i++) {
        waiting_time[i] = waiting_time[i - 1] + arr[i - 1][2];
        total_waiting_time += waiting_time[i];
        printf("%d\n", waiting_time[i]);
    }

    avg_time = (float)total_waiting_time / n;
    printf("Average Waiting Time: %f\n", avg_time);
    printf("Non-Preemptive Completion Time:\n");
    completion_time(arr, n);
}

void sjf_preemptive(int arr[][3], int n) {
    // Sorting processes based on arrival time
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j][1] > arr[j + 1][1]) {
                for (int k = 0; k < 3; k++) {
                    swap(&arr[j][k], &arr[j + 1][k]);
```

```
            }
        }
    }
}

int remaining[n];
int waiting_time[n];
for (int i = 0; i < n; i++) {
    remaining[i] = arr[i][2]; // Initialize remaining times
    waiting_time[i] = 0; // Initialize waiting times
}

int t = 0;  // Current time
int count = 0;  // Number of processes completed
int completion_time[n]; // To store completion times

while (count < n) {
    int min_remaining = 1000000;
    int min_index = -1;

    // Find the process with the shortest remaining time that has arrived
    for (int i = 0; i < n; i++) {
        if (arr[i][1] <= t && remaining[i] > 0 && remaining[i] < min_remaining) {
            min_remaining = remaining[i];
            min_index = i;
        }
    }

    if (min_index == -1) {
        // No process is currently available to run
        t++;
        continue;
    }

    // Execute the process with the shortest remaining time
    remaining[min_index]--;

    // If the process is completed
    if (remaining[min_index] == 0) {
        count++;
        completion_time[min_index] = t + 1;  // Completion time
        waiting_time[min_index] = completion_time[min_index] - arr[min_index][1] -
arr[min_index][2];
    }
    t++; // Increment time
}

// Output results
```

```
    printf("Process ID\tWaiting Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\n", arr[i][0], waiting_time[i]);
    }

    printf("\nProcess ID\tCompletion Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\n", arr[i][0], completion_time[i]);
    }
}

void round_robin(int arr[][3], int n) {
    // Sorting processes based on arrival time
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j][1] > arr[j + 1][1]) {
                for (int k = 0; k < 3; k++) {
                    swap(&arr[j][k], &arr[j + 1][k]);
                }
            }
        }
    }

    int remaining[n];
    int waiting_time[n];
    for (int i = 0; i < n; i++) {
        waiting_time[i] = 0;
        remaining[i] = arr[i][2];
    }

    int t = 0;  // Current time
    int count = 0;  // Number of processes completed
    int completion_time[n];

    int tq;
    printf("Enter The Time Quantum: ");
    scanf("%d", &tq);

    while (count < n) {
        int found = 0; // Flag to check if a process is found
        for (int i = 0; i < n; i++) {
            if (arr[i][1] <= t && remaining[i] > 0) {
                found = 1; // Process found
                if (remaining[i] > tq) {
                    t += tq;
                    remaining[i] -= tq;
                } else {
                    t += remaining[i];
```

```
                completion_time[i] = t;
                waiting_time[i] = t - arr[i][1] - arr[i][2];
                remaining[i] = 0;
                count++;
            }
        }
    }
    if (!found) {
        t++; // Increment time if no process is found
    }
}

printf("Process ID\tWaiting Time\n");
for (int i = 0; i < n; i++) {
    printf("%d\t\t%d\n", arr[i][0], waiting_time[i]);
}

printf("\nProcess ID\tCompletion Time\n");
for (int i = 0; i < n; i++) {
    printf("%d\t\t%d\n", arr[i][0], completion_time[i]);
}
}

void print(int arr[][3], int n) {
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%d\t", arr[i][j]);
    }
    printf("\n");
}
}

int main() {
int n;
printf("Enter number of processes: ");
scanf("%d", &n);

int arr[n][3];
printf("Enter process ID, arrival time, and burst time for each process:\n");
for (int i = 0; i < n; i++) {
    printf("Process %d: ", i + 1);
    for (int j = 0; j < 3; j++) {
        scanf("%d", &arr[i][j]);
    }
}

print(arr, n);
printf("********** SJF Non-Preemptive **********\n");
```

```
    sjf_non_preemptive(arr, n);

    printf("\n*************** SJF Preemptive **************\n");
    sjf_preemptive(arr, n);

    printf("\n*************** Round Robin **************\n");
    round_robin(arr, n);

    return 0;
}
```

Output

```
Enter number of processes: 4
Enter process ID, arrival time, and burst time for each process:
Process 1: 1
0
8
Process 2: 2
1
4
Process 3: 3
2
9
Process 4: 4
3
5
1       0       8
2       1       4
3       2       9
4       3       5
************* SJF Non-Preemptive ************
Process with earliest arrival time:
Process ID: 1, Arrival Time: 0, Burst Time: 8
Process ID      Arrival Time    Burst Time
1               0               8
2               1               4
4               3               5
3               2               9
Non-Preemptive Waiting Time:
0
8
12
17
Average Waiting Time: 9.250000
Non-Preemptive Completion Time:
2               12
4               17
```

```
3               26

********** SJF Preemptive **********
Process ID      Waiting Time
1               9
2               0
3               15
4               2

Process ID      Completion Time
1               17
2               5
3               26
4               10

********** Round Robin **********
Enter The Time Quantum: 2
Process ID      Waiting Time
1               15
2               7
3               15
4               13

Process ID      Completion Time
1               23
2               12
3               26
4               21



Enter number of processes: 5
Enter process ID, arrival time, and burst time for each process:
Process 1: 10
0
6
Process 2: 2
2
2
Process 3: 3
4
1
Process 4: 46
6
7
Process 5: 5
8
5
```

1 0 6
2 2 2
3 4 1
4 6 7
5 8 5
********** SJF Non-Preemptive **********
Process with earliest arrival time:
Process ID: 1, Arrival Time: 0, Burst Time: 6
Process ID Arrival Time Burst Time
1 0 6
3 4 1
2 2 2
5 8 5
4 6 7
Non-Preemptive Waiting Time:
0
6
7
9
14
Average Waiting Time: 7.200000
Non-Preemptive Completion Time:
3 7
2 9
5 14
4 21

********** SJF Preemptive **********
Process ID Waiting Time
1 3
2 0
3 0
4 8
5 1

Process ID Completion Time
1 9
2 4
3 5
4 21
5 14

********** Round Robin **********
Enter The Time Quantum: 2
Process ID Waiting Time
1 7
2 0
3 0

4	8
5	7

Process ID	Completion Time
1	13
2	4
3	5
4	21
5	20